Git

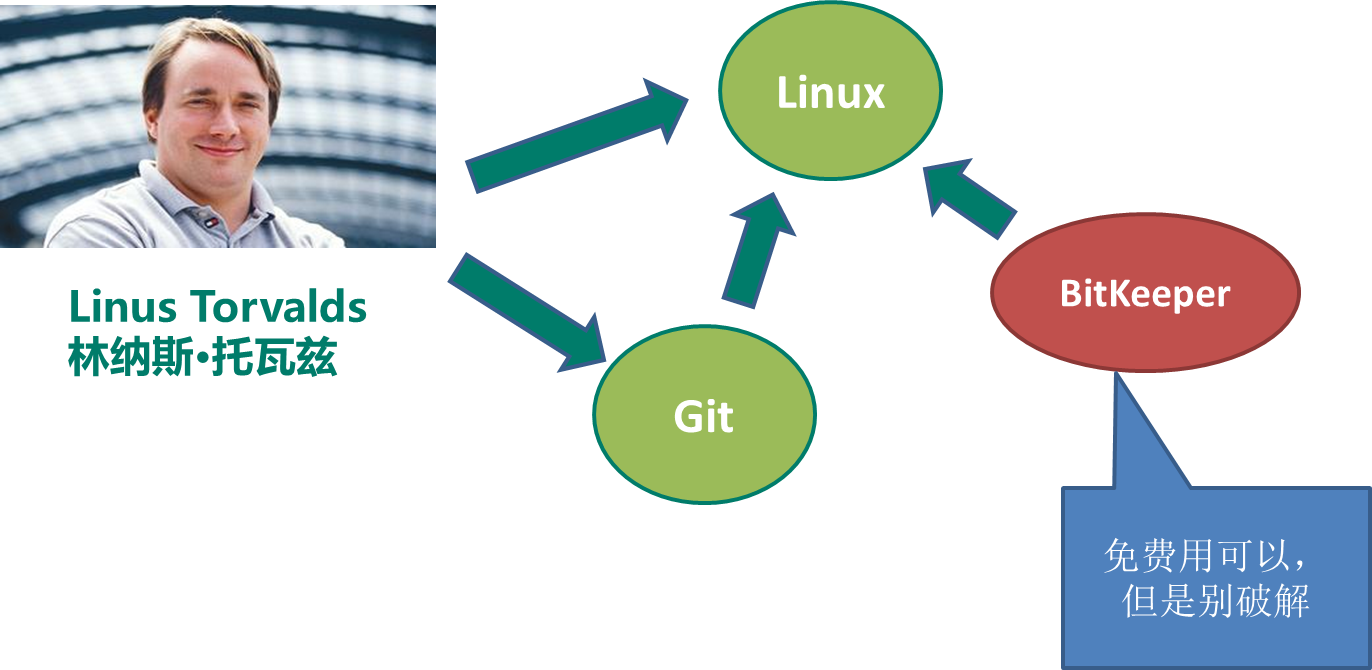
尚硅谷大数据

版本：V2.0

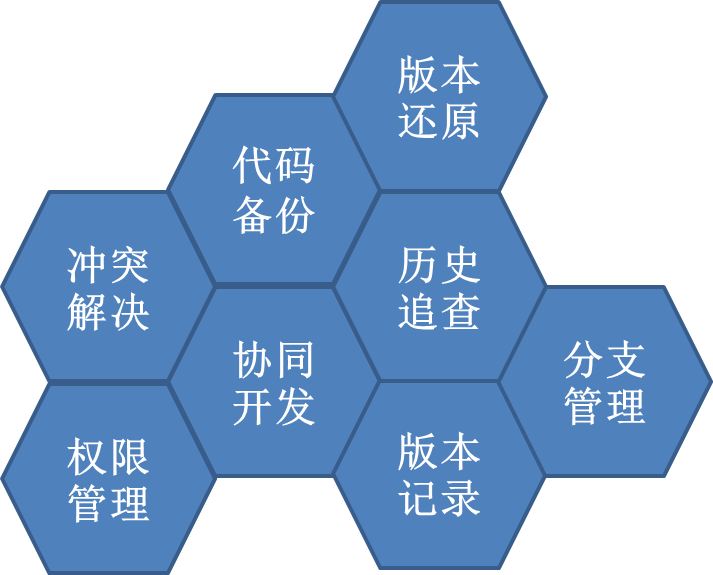
# 一、Git简介

## 1. 是什么

Git是目前世界上最先进的分布式版本控制系统！



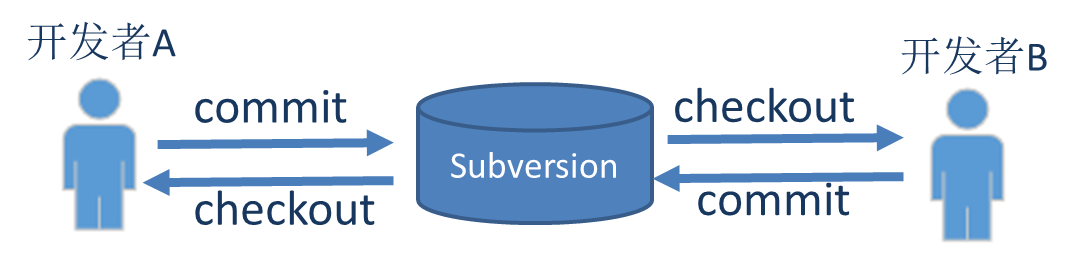
## 2. 能干吗



## 3. 集中式版本管理

**经典产品**： CVS、VSS、SVN

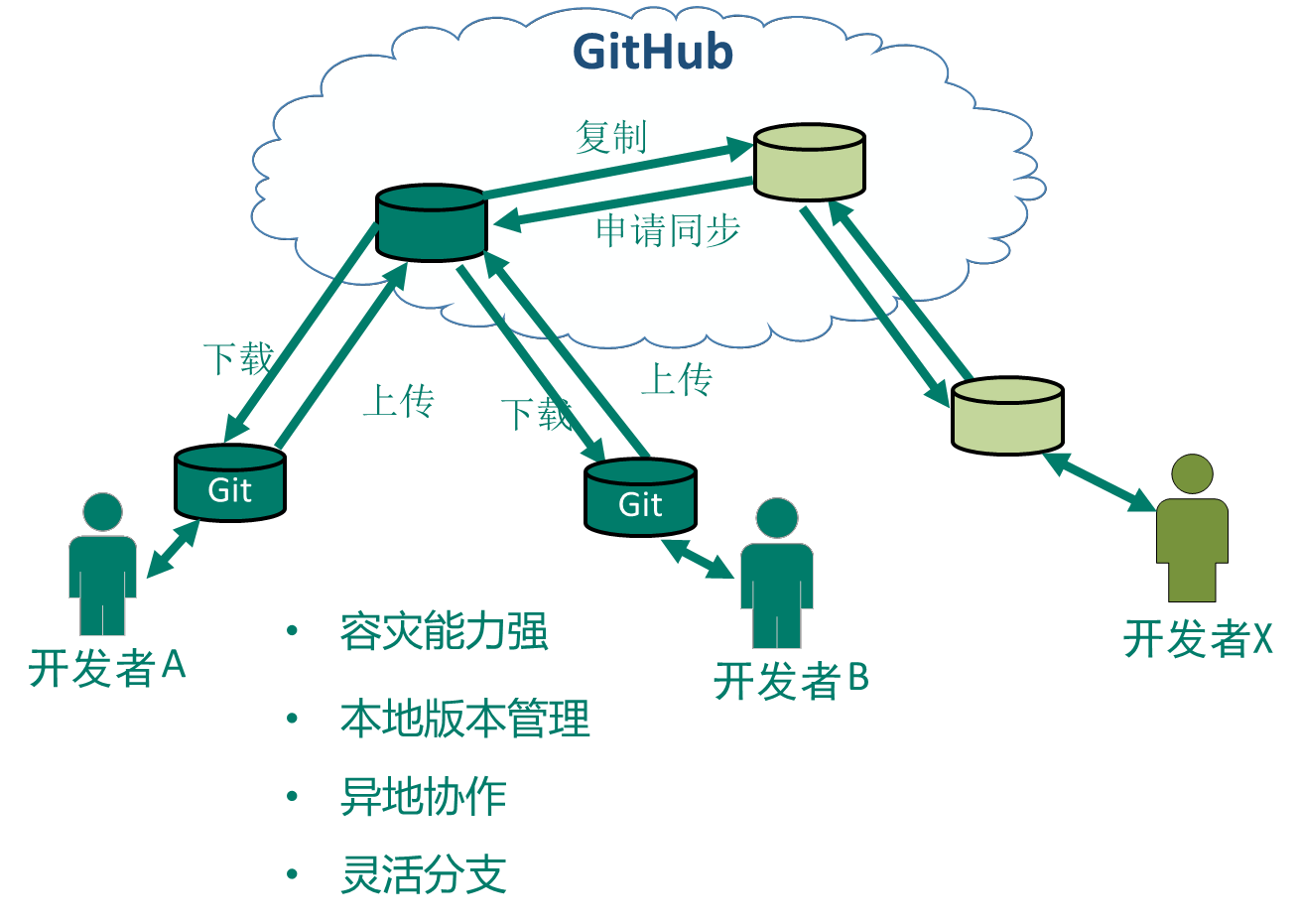
**特点**：由中央仓库统一管理，结构简单，上手容易！



**不足**：

* + - * 版本管理的服务器一旦崩溃，硬盘损坏，代码如何恢复？
      * 程序员上传到服务器的代码要求是完整版本，但是程序员开发过程中想做小版本的管理，以便追溯查询，怎么破？
      * 系统正在上线运行，时不时还要修改bug，要增加好几个功能要几个月，如何管理几个版本？
      * 如何管理一个分布在世界各地、互不相识的大型开发团队？

**解决**：



# 二、Git的安装

Git官网：https://git-scm.com/

## 1. 相关工具

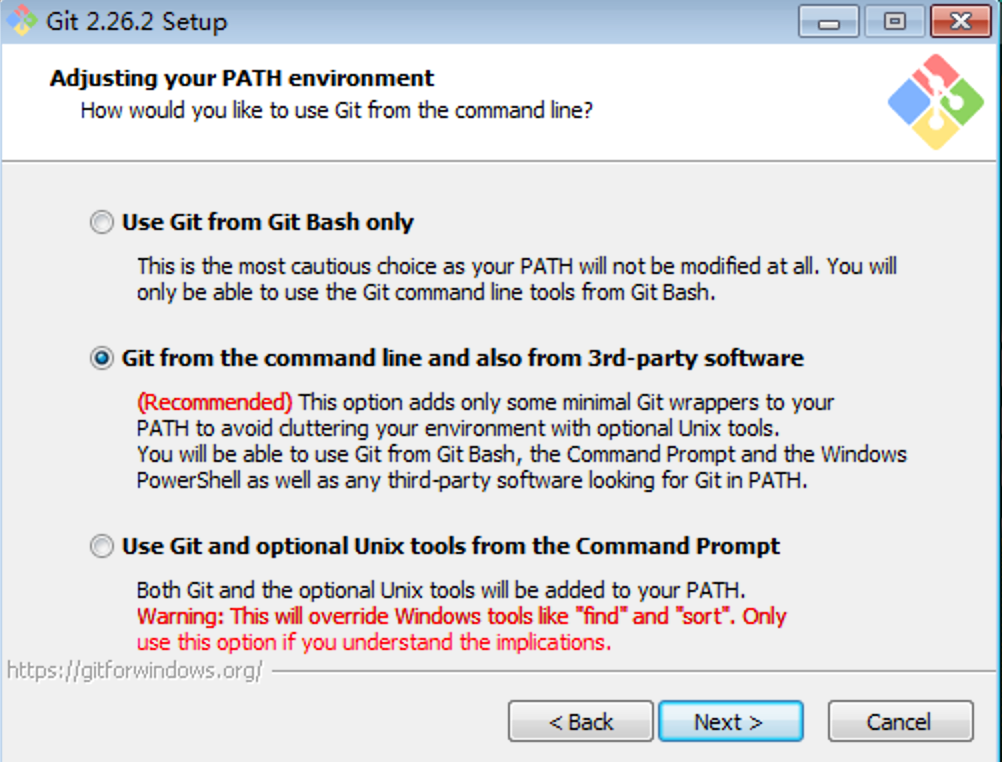
* 命令行工具：Git for windows

下载地址：https://git-for-windows.github.io/

* ijdea插件： idea 自带
* GitHub网站:http://www.github.com

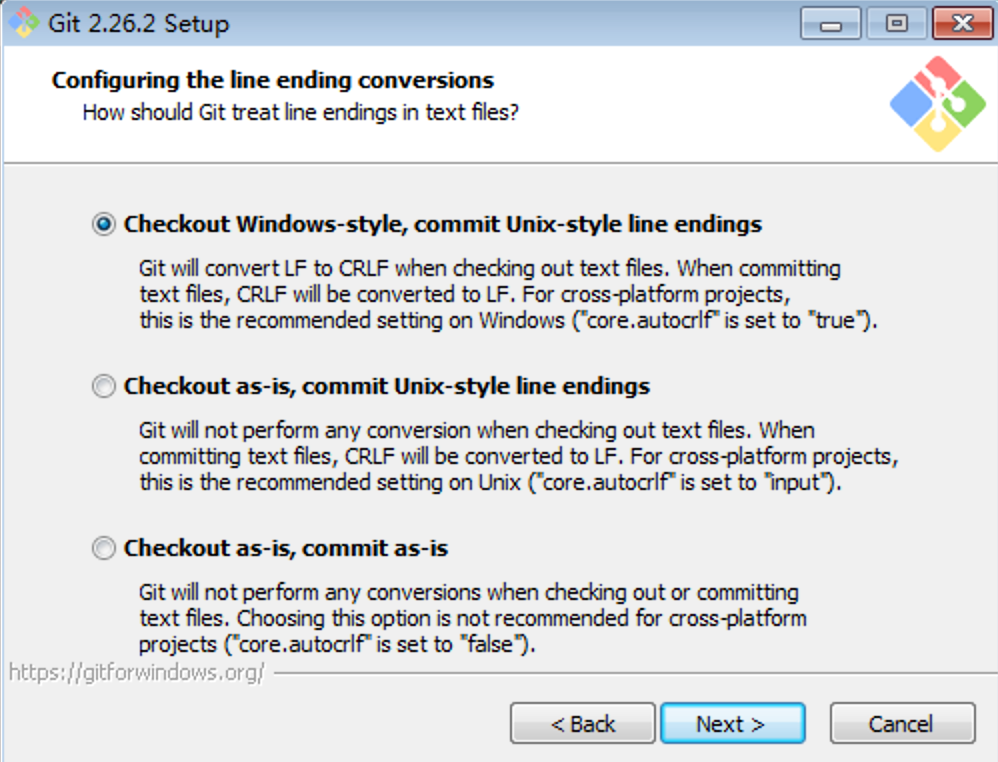
## 2. 安装流程

一直下一步，直到：



选择Git命令的执行环境，这里推荐选择第一个，就是单独用户Git自己的命令行窗口。

不推荐和windows的命令行窗口混用。



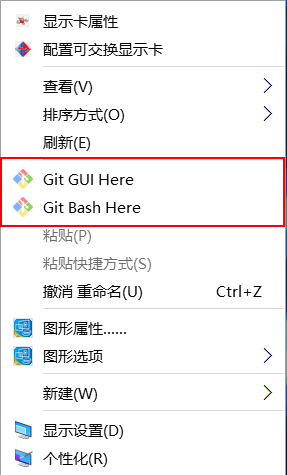
在“Configuring the line ending conversions”选项中，

第一个选项：如果是跨平台项目，在windows系统安装，选择；

第二个选项：如果是跨平台项目，在Unix系统安装，选择；

第三个选项：非跨平台项目，选择。

安装成功后：右键菜单，会多出以下选项卡：

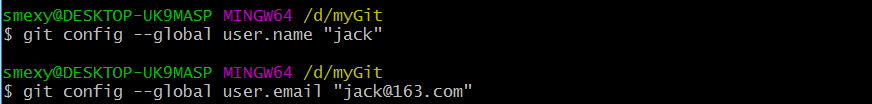


## 3. 设置Git账户

|  |  |
| --- | --- |
| 命令 | 含义 |
| git config --list | 查看所有配置 |
| git config --list --show-origin | 查看所有配置以及所在文件位置 |
| git config --global user.name xxx | 设置git用户名 |
| git config --global user.email xxx | 设置git邮箱 |
| git init | 初始化本地库 |
| git config core.autocrlf false | 取消换行符转换的warning提醒 |

上述配置可以在用户家目录/.gitconfig文件中查看配置结果！

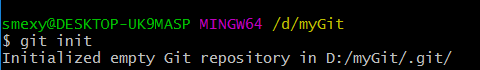
示例：



## 4. 初始化

①新建一个本地仓库，也就是一个文件夹。

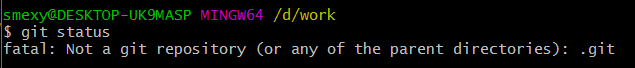
②执行git init命令



之后，可以看到，本地仓库中，多了一个.git的隐藏文件夹



只有在有.git目录的工作空间中，才可以执行git命令！否则，会报错：



# 三、常用Git命令

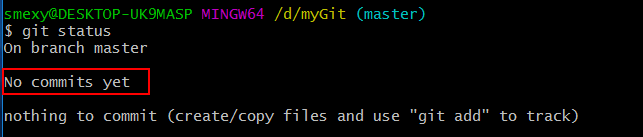
## 1. 新建文件并提交到本地库

### 1.1 常用命令

|  |  |
| --- | --- |
| 命令 | 作用 |
| git status | 查看本地库的状态(git status -s 简化输出结果) |
| git add [file] | 多功能命令: 1. 开始跟踪新文件 2. 把已跟踪的文件添加到暂存区 3. 合并时把有冲突的文件标记为已解决状态 |
| git commit –m “xxx” [file] | 将暂存区的文件提交到本地库,-m 后面为修改的说明 |

### 1.2 实验步骤

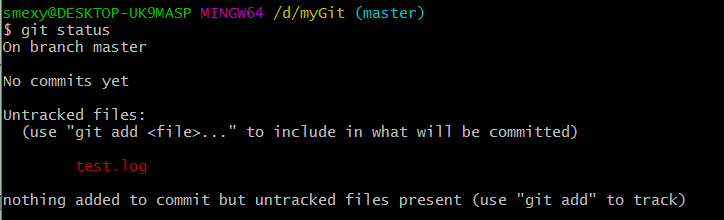
#### ①查看本地库状态，空空如也



#### ②新建一个文件

vim test.log 或 touch test.log并编辑

此时再次查看git本地库的状态：

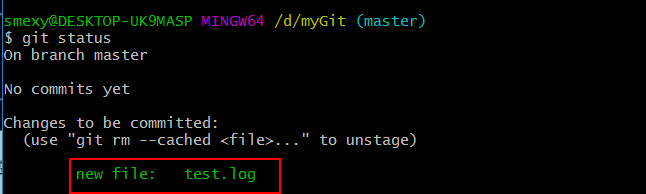


说明检测到了test.log文件，但是还无法追踪，建议使用add命令添加！

#### ③将新建的文件，提交到暂存区：git add



#### ④此时再次查看，发现暂存区已经有了文件

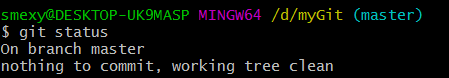


⑤将文件从暂存区提交到本地库



此时，会要求为刚刚的提交作一个说明！

也可以直接使用git commit –m “xxxx” [file]提交！

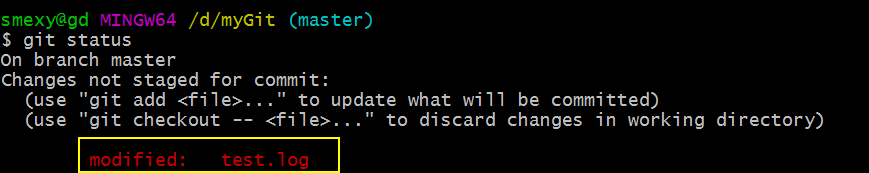


## 2. 将修改的文件提交到本地库

#### ①修改文件



#### ②此时查看本地库的状态



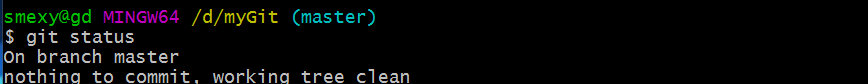
发现已经追踪到了，文件发生了修改！

#### ③提交修改

先add，再commit!



#### ④此时再查看本地库的状态



## 3. 忽略文件

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。

通常都是些自动生成的文 件，比如日志文件，或者编译过程中创建的临时文件等。

在这种情况下，我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件的模式。

一个.gitignore 的案例:

# 忽略所有的

.a 文件 \*.a

# 但跟踪所有的 lib.a，即便你在前面忽略了 .a 文件

!lib.a

# 只忽略当前目录下的 TODO 文件，而不忽略 subdir/TODO. 不递归的忽略

/TODO

# 忽略任何目录下名为 build 的文件夹 递归的忽略

build/

# 忽略 doc/notes.txt，但不忽略 doc/server/arch.txt

doc/\*.txt

# 忽略 doc/ 目录及其所有子目录下的 .pdf 文件

doc/\*\*/\*.pdf

## 4. 版本切换

### 4.1 常用命令

|  |  |
| --- | --- |
| 命令 | 作用 |
| git log | 以完整格式查看本地库状态(查看提交历史) |
| git log --pretty=oneline | 以单行模式查看本地库状态 |
| git reset --hard HEAD^ | 回退一个版本 |
| git reset --hard HEAD~n | 回退N个版本 |
| git reflog | 查看所有操作的历史记录 |
| git reset --hard [具体版本号，例如：1f9a527等] | 回到（回退和前进都行）指定版本号的版本， |
| git checkout -- [file] | 从本地库检出最新文件覆盖工作区的文件(文件还没有提交到暂存区, 否则无效) |
| git reset [file] 或者 git restore –staged [file] | 从暂存区撤销文件 |
| git restore <file> | 放弃在工作区的修改(还没有提交到暂存区) |
| git rm --cache [file] | 撤销对文件的跟踪. |

注意:

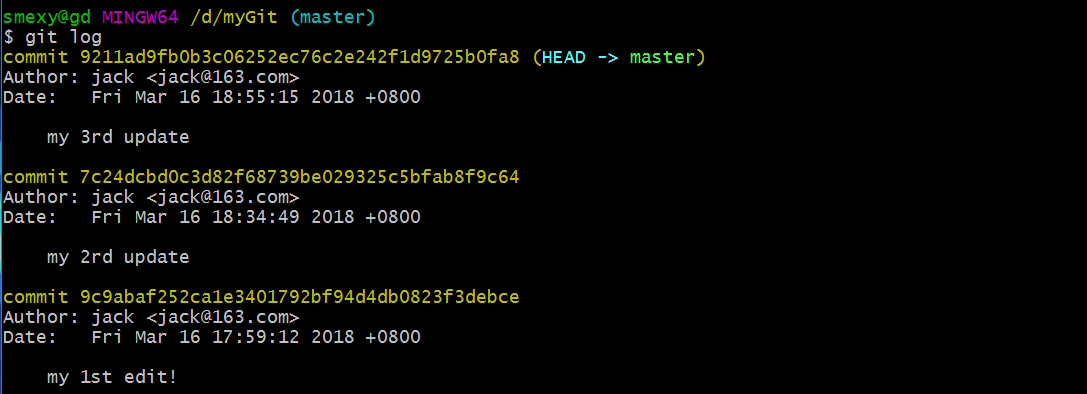
1. 在 Git 中任何已提交的东西几乎总是可以恢复的
2. 任何你未提交的东西丢失后很可能再也找不到了

### 4.2 回退实验步骤

多次更新之后，希望回退到之前的修改版本

#### ①查看更新日志

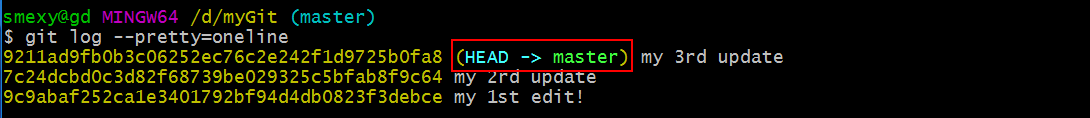
git log



如果日志过于凌乱，可以使用一个简单格式查看

#### ②单行简单格式查看：

git log --pretty=oneline



#### ③回退到历史版本

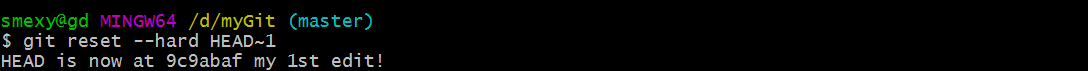
第一种： git reset --hard HEAD^

回退到上一个版本，如果想回退多个版本，就多加几个^



第二种：git reset --hard HEAD~n

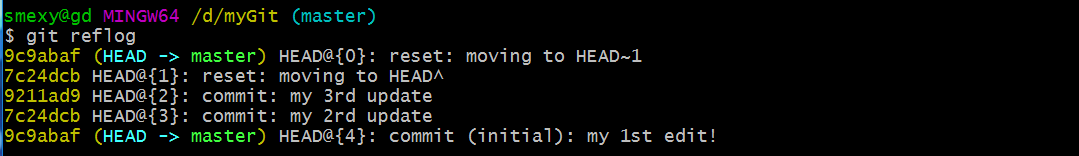
回退多个版本



### 3.3 前进实验步骤

#### ①查看所有操作的历史记录

git reflog



#### ②回退到指定的操作

git reset --hard [具体版本号，例如：1f9a527等]



### 4.4 文件恢复

#### 4.4.1 从本地库检出恢复

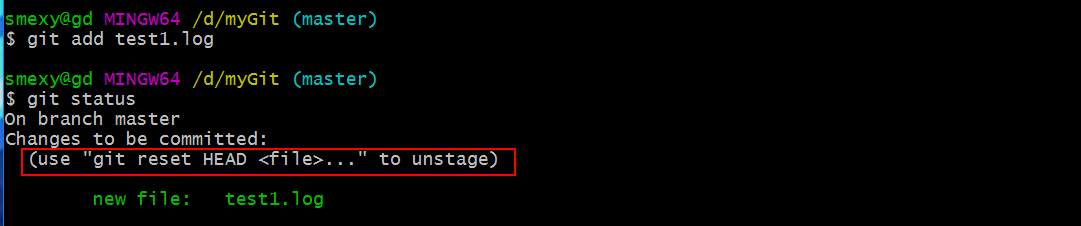
git checkout -- [file]



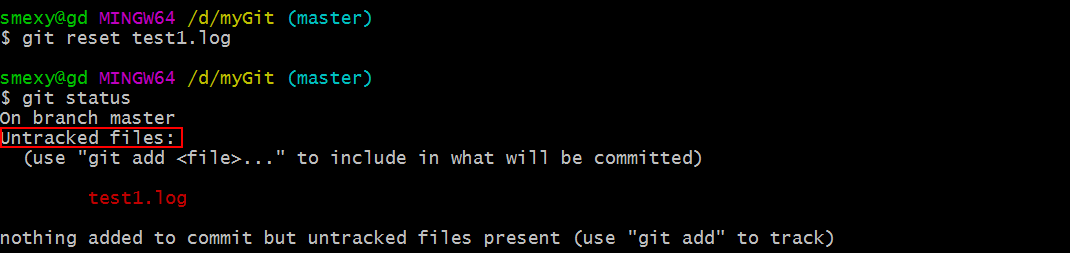
#### 4.4.2 从暂存区撤回

当add一个文件到暂存区后，可以执行commit操作，也可以执行reset撤回操作！

①添加完成后，查看暂存区情况



②git reset HEAD [file]



### 4.5 删除操作

如果删除一个已经在版本库中的文件！

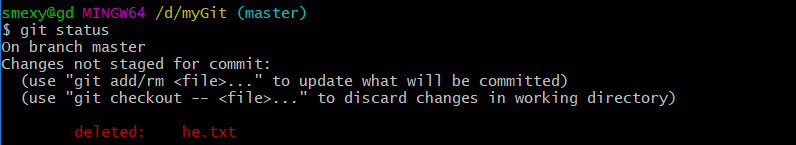
①删除文件



此时查看，发现虽然在windows的文件系统中已经没有了：

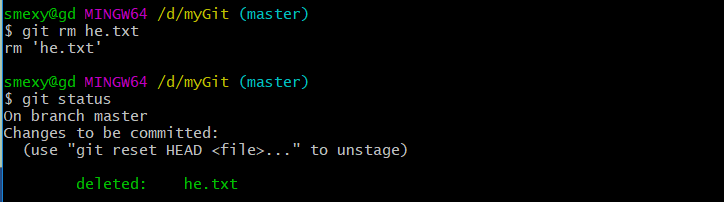


但是，使用git status查看版本库的情况，发现：



提示，使用git add/rm <file>命令来更新此次操作！

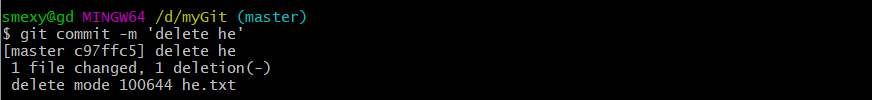
之后，再次查看



发现变成了绿色，提交已经成功！

②提交更新

然后执行，提交操作：



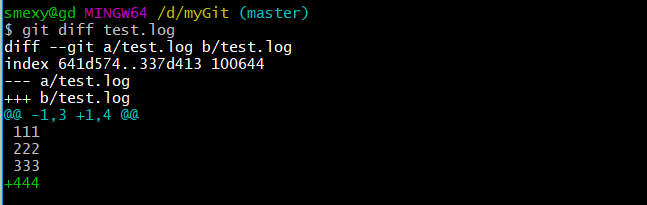
再次查看git status，发现完成了此次操作！

### 4.6 比较文件

减号和加号分别代表两个对比的文件！

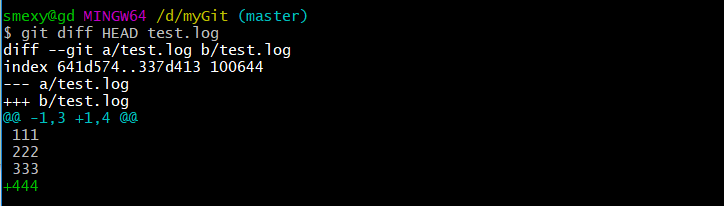
#### 4.6.1 将工作区中的文件和暂存区进行比较

$ git diff <file>



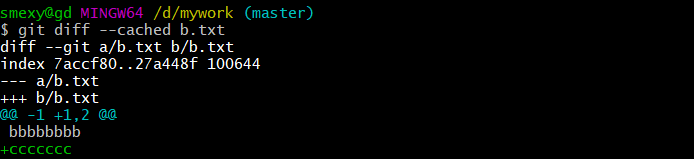
#### 4.6.2 将工作区中的文件和本地库当前版本进行比较

$ git diff HEAD <file>



#### 4.6.3 查看暂存区和本地库最新提交版本的差别

git diff --cached <file>



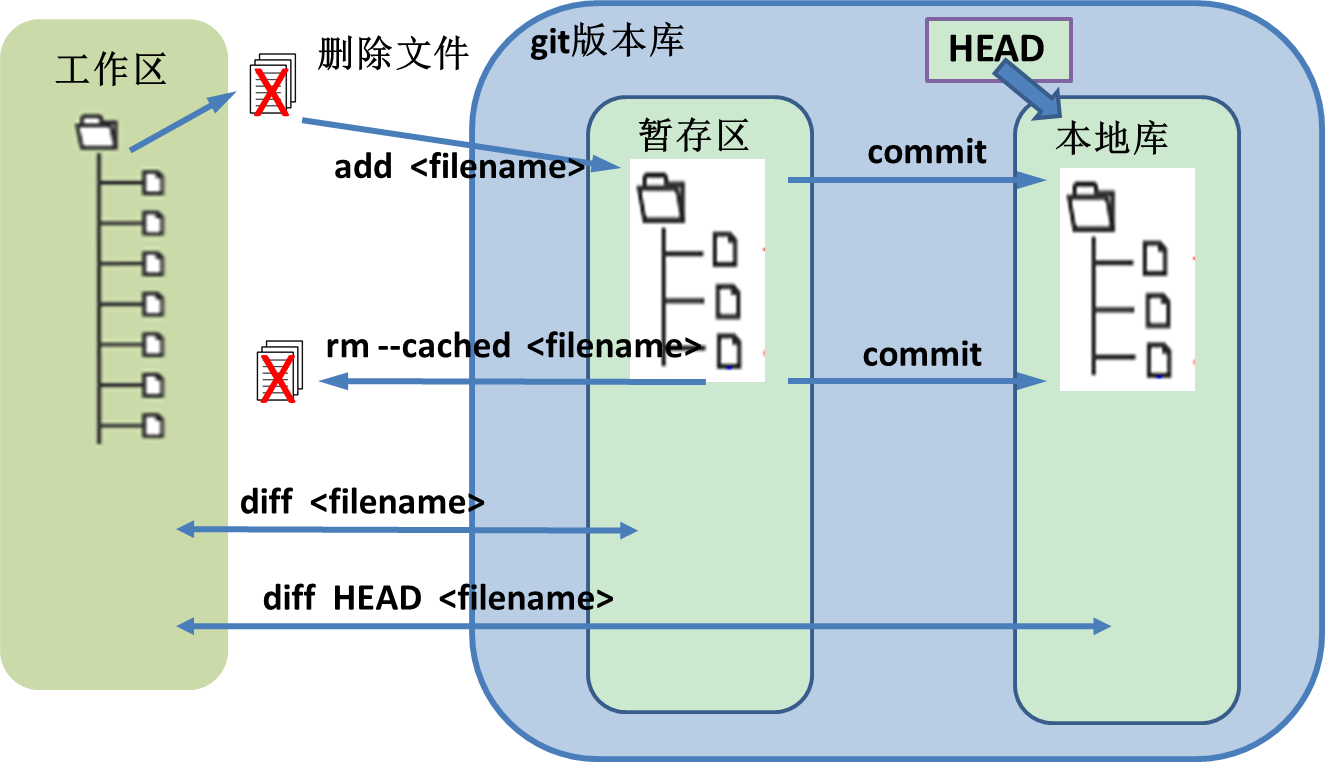
# 四、Git的工作机制

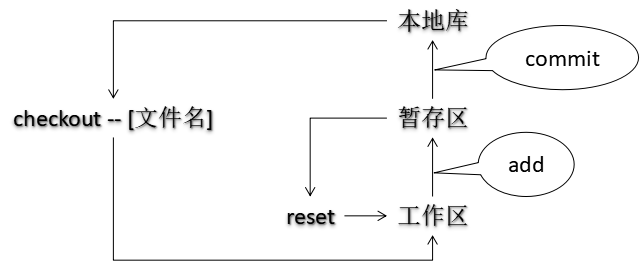
## 1. 三区

工作区(Working Directory):就是你电脑本地硬盘目录

本地库(Repository):工作区有个隐藏目录.git，它就是Git的本地版本库

暂存区(stage):一般存放在"git目录"下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。





红色状态代表在工作区，尚未添加到暂存区！

绿色状态代表当前文件已经添加到了暂存区，但是尚未添加到本地库！

暂存区空了，代表在所有文件已经添加到本地库；

## 2. .git目录



# 五、分支操作

## 1. 常用命令汇总

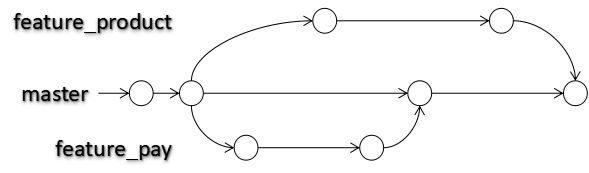
|  |  |
| --- | --- |
| 命令 | 描述 |
| git branch [分支名] | 创建分支 |
| git branch -v | 查看分支,可以使用-v参数查看详细信息 |
| git checkout [分支名] | 切换分支 |
| git merge [分支名] | 合并分支；  将merge命令中指定的分支合并到当前分支上  例如：如果想将dev分支合并到master分支，那么必须在master分支上执行merge命令 |
| git branch –d[分支名] | 删除分支 |
| git checkout –b [分支名] | 新建并切换到当前分支 |
| git log --oneline --decorate --graph --all | 它会输出你的提交历史、各个分支的指向以及项目的分支分叉情况 |

## 2. 分支的概念

不使用分支，就是人与人之间协作；

使用分支，就是小组与小组之间的协作；

从主干中拉取分支，开发完成，将工作，合并到主干。



## 3. 分支常用操作

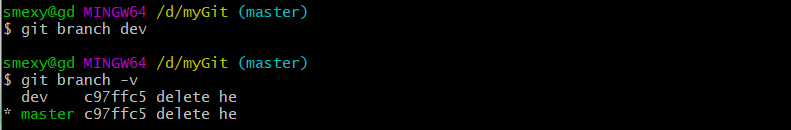
### 3.1 查看分支

git branch -v



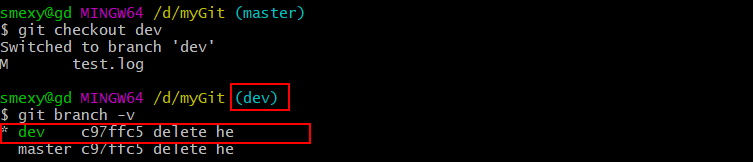
### 3.2 新建分支

git branch [分支名]



### 3.3 切换分支

git checkout [分支名]

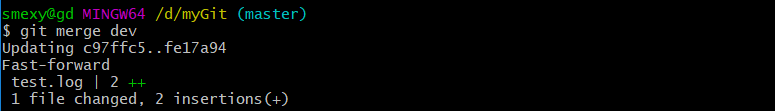


### 3.4 合并分支

git merge [分支名]

如果在dev分支上面，对一个文件做了修改，这个时候master分支上面对应的文件是没有修改信息的。

因此需要将分支合并！



注意：必须切换到master分支！

### 3.5 删除分支

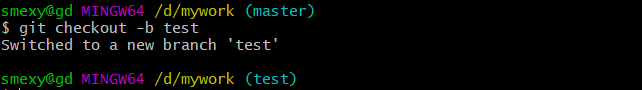
git branch –d [分支名]

注意：必须切换到master，才能删除，不能自杀！



### 3.6 新建切换

git checkout –b [分支名]



## 4. 处理分支冲突

待完成

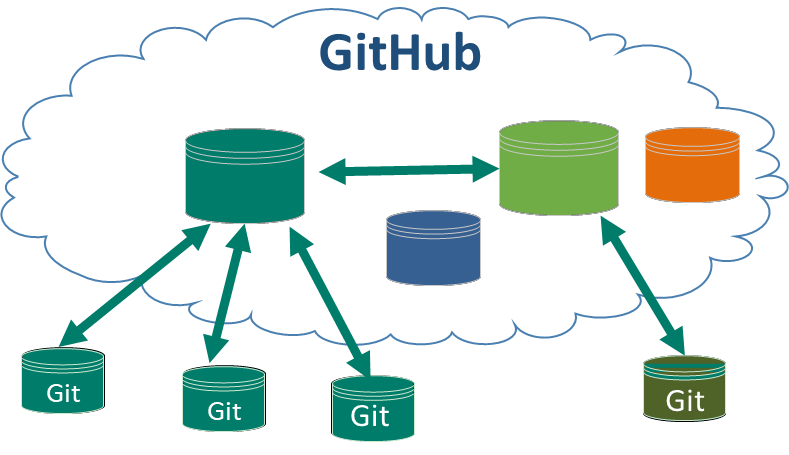
编辑冲突的文件，把“>>>>>>>>>”、“<<<<<<”和“========”等这样的行删除，编辑至满意的状态，提交。

提交的时候注意：git commit命令不能带文件名。

# 六、GitHub

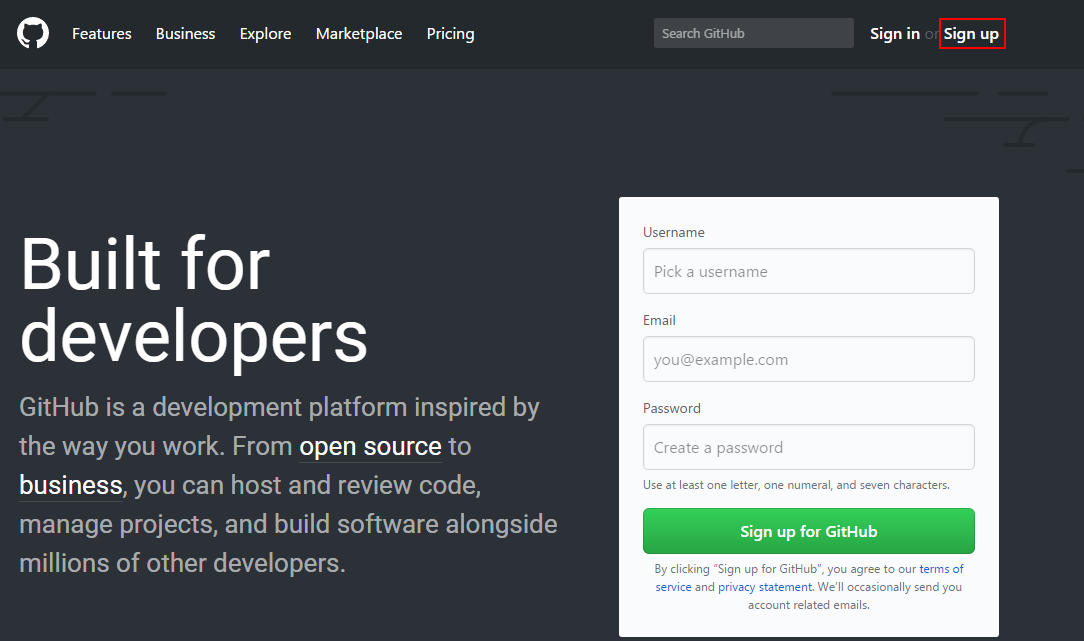
## 1. 简介

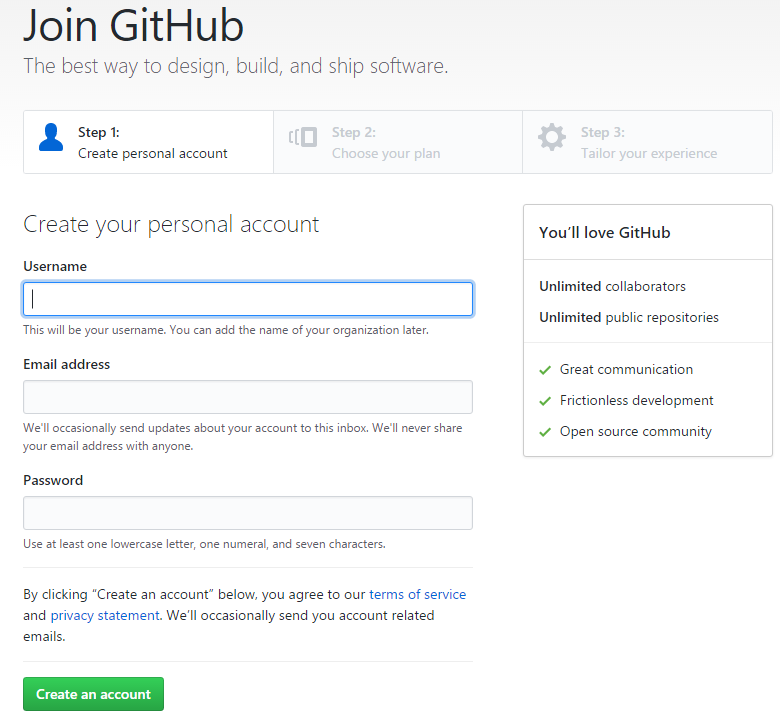
GitHub是一个Git项目托管网站,主要提供基于Git的版本托管服务。



网址：**： https://github.com/**

## 2. 注册





## 3. 简单操作

### 3.1 本地库联通GitHub

①查看本地是否配置了密钥



只需要在第一次配置就可以了，如果存在就不需要额外生成密钥

②生成密钥

命令：ssh-keygen

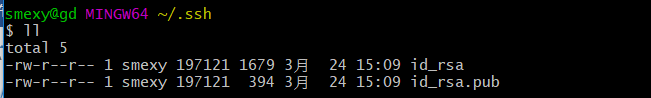
查看用户名：



生成密钥：



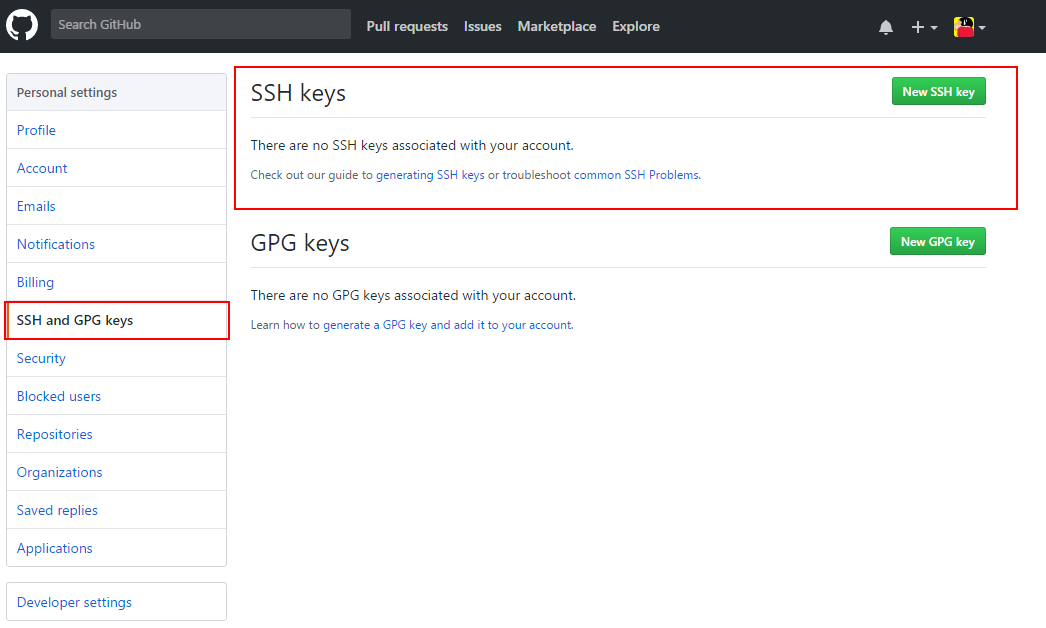
生成的密钥分为私钥和公钥，会保存在用户家目录的.ssh文件夹中。



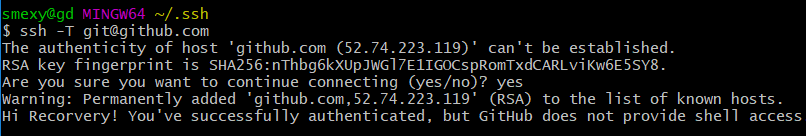
其中，id\_rsa是私钥，id\_rsa.pub是公钥。

将公钥告诉github账户，相当于约定一个接头暗号！

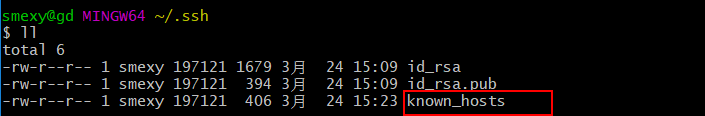
进入github的账户，



测试本地仓库和github的联通性：ssh -T git@github.com



.ssh文件夹中会多一个文件known\_hosts，其中记录了连接的github的ip账号。



### 3.2 push

**本地库推送到GitHub**

①准备本地库

②在GitHub上创建一个仓库

③增加远程地址

git remote add <远端代号> <远端地址>

<远端代号> 是指远程链接的代号，一般直接用origin作代号，也可以自定义；

<远端地址> 默认远程链接的url；



④本地库推送到远程库

git push -u <远端代号> <本地分支名称>

<远端代号> 是指远程链接的代号；

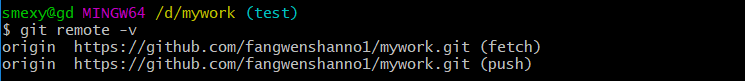
<分支名称> 是指要提交的分支名字，比如master；

我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

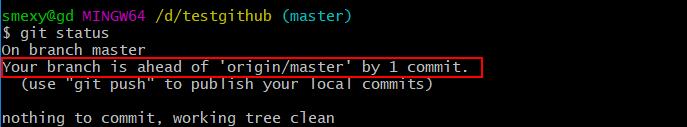


⑤查看远程分支

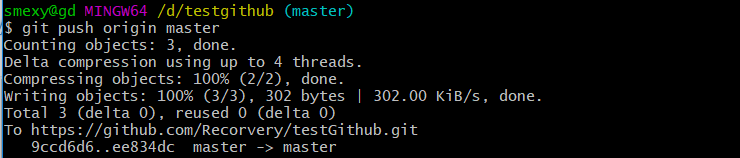
git remote –v



⑥如果此时本地库做了修改，那么查看：



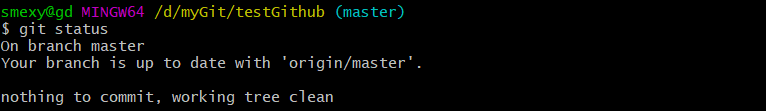
发现当前的本地库比远程库新增了一些内容。



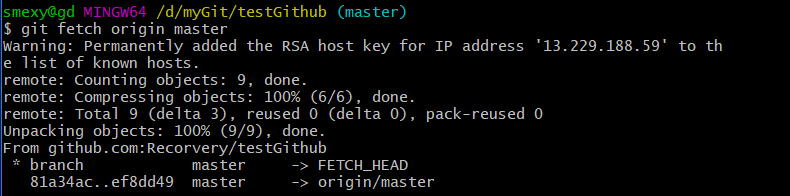
### 3.3 fetch

fetch命令可以从远程库获取更新，但是并不合并！

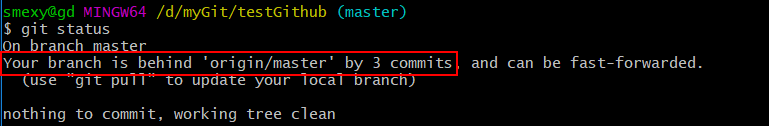
远程库已经更新，此时查看本地库的状态：



执行git fetch 命令，查看远程库的更新；



再查看git status



### 3.4 pull

**本地库抓取远程库**

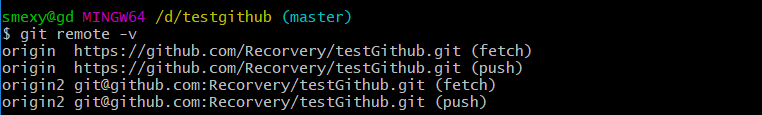
如果远程库的版本新于当前库，那么此时为了使当前库和远程库保持一致，可以执行pull命令

git pull <远端代号> <远端分支名>;

例 git pull origin master



查看有哪些本地库：



### 3.5 clone

实际的情况可能是刚开始做项目的时候，需要从远程库将项目先整到本机。

执行命令：git clone <远端地址> <新项目目录名>

<远端地址> 是指远程链接的地址;

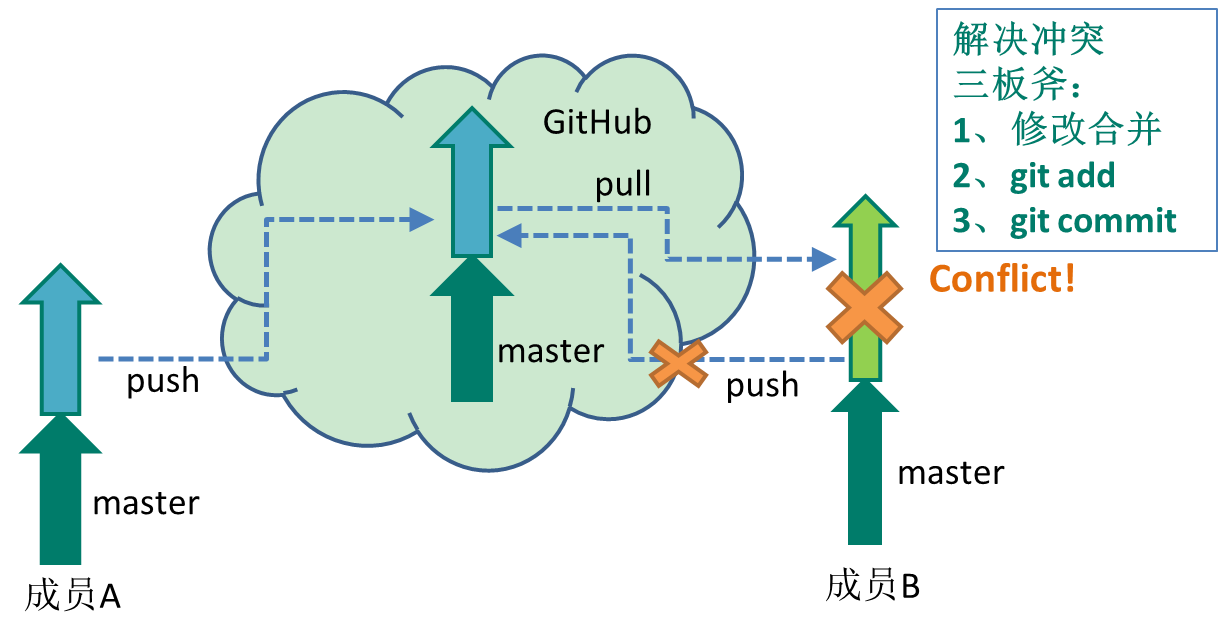
<项目目录名> 是指为克隆的项目在本地新建的目录名称，可以不填，默认是GitHub的项目名;

命令执行完后，会自动为这个远端地址建一个名为origin的代号。

例 git clone https://github.com/user111/Helloworld.git hello\_world



### 3.6 解决冲突



此时，远程库比本地库的版本要新，因此，本地库执行拉取操作！git pull!

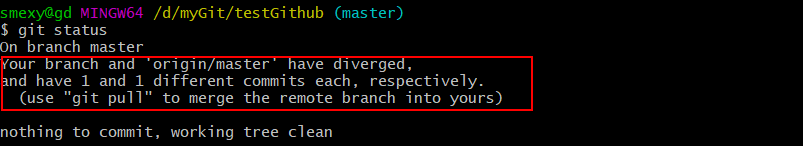


冲突原因：本地库的版本和远程库的版本都做了更新！

在推送之前，执行git fetch，发现远程库和本地库有版本变化。

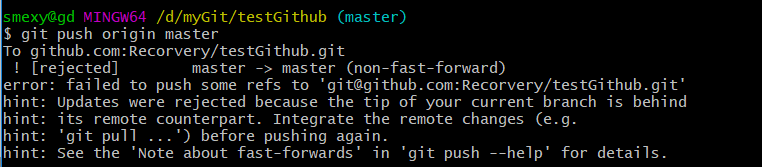


查看git status，



此时，建议先pull，将远程库的版本和本地库的版本merge后再push！

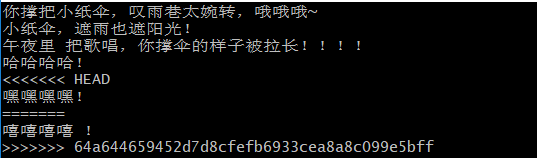
如果强行push,报错！



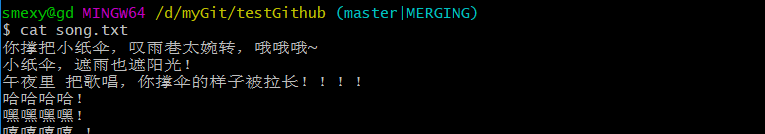
解决办法：先pull，pull的时候会发现报错冲突



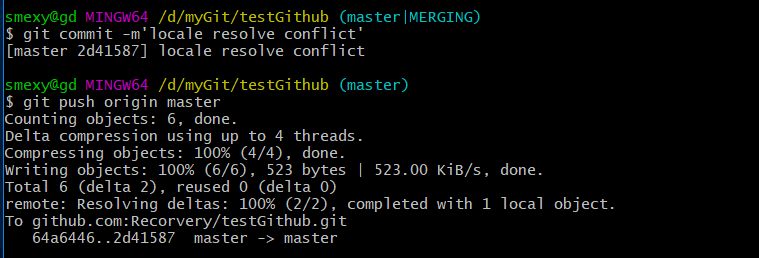
因此开始解决冲突，编辑冲突文件！



编辑完成：



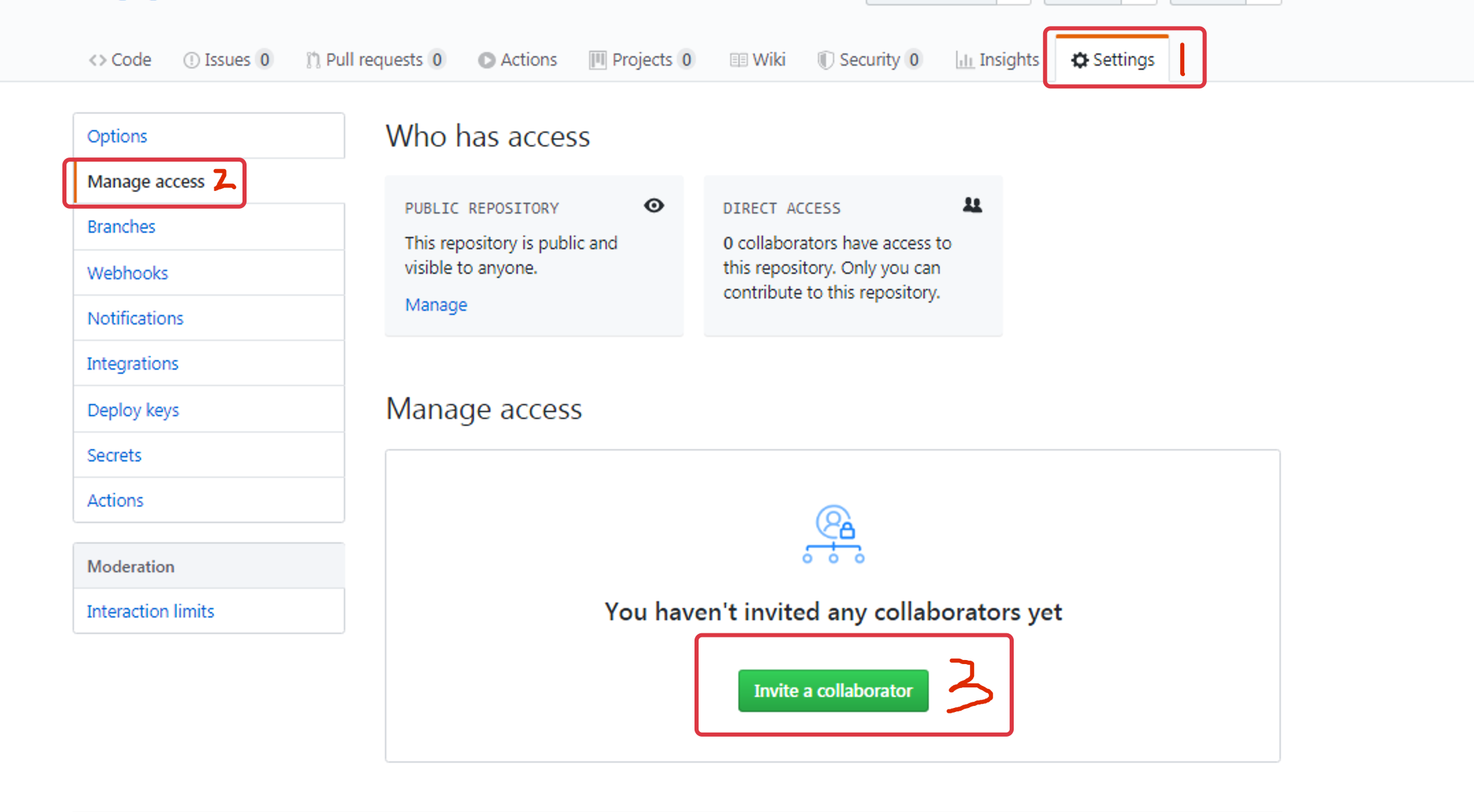
之后再推：



总结：pull----merge-----push

### 3.7 邀请成员

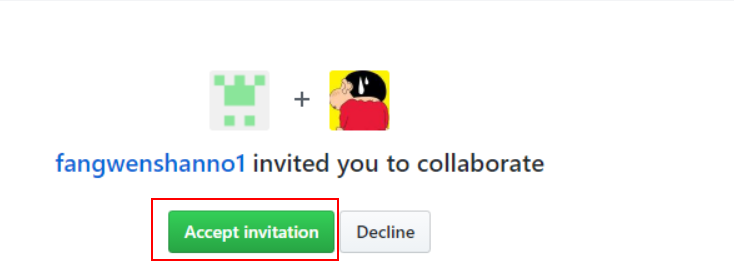
①在相关的项目设置中, 搜索要邀请的成员：



③复制邀请连接，手动告知被邀请者：



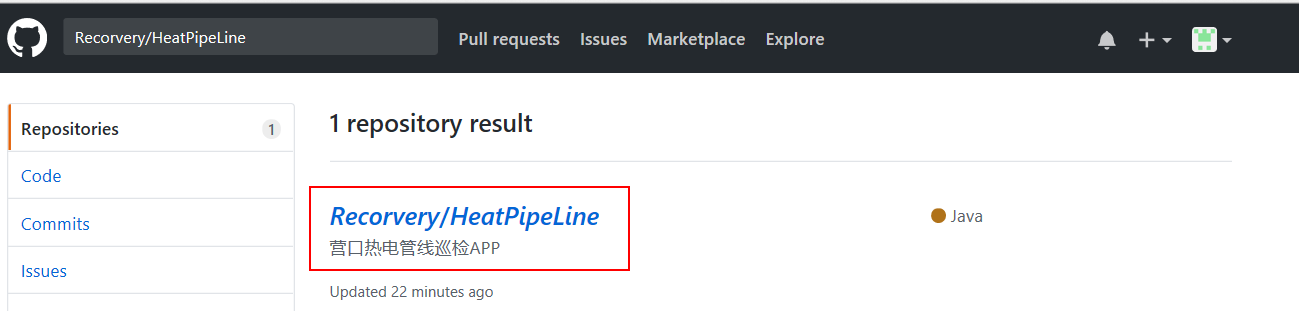
④被邀请者，登录自己的账号后，点击连接，接收邀请



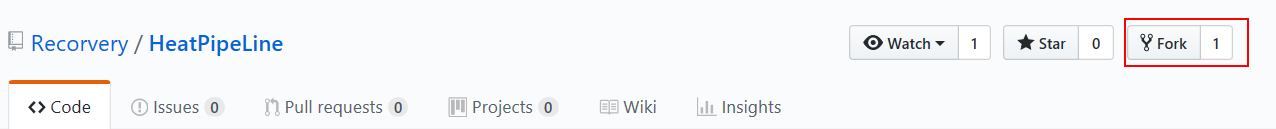
### 3.7 fork

如果其他人，搜索到了你的项目，想对其做一些编辑时，必须先执行fork操作。

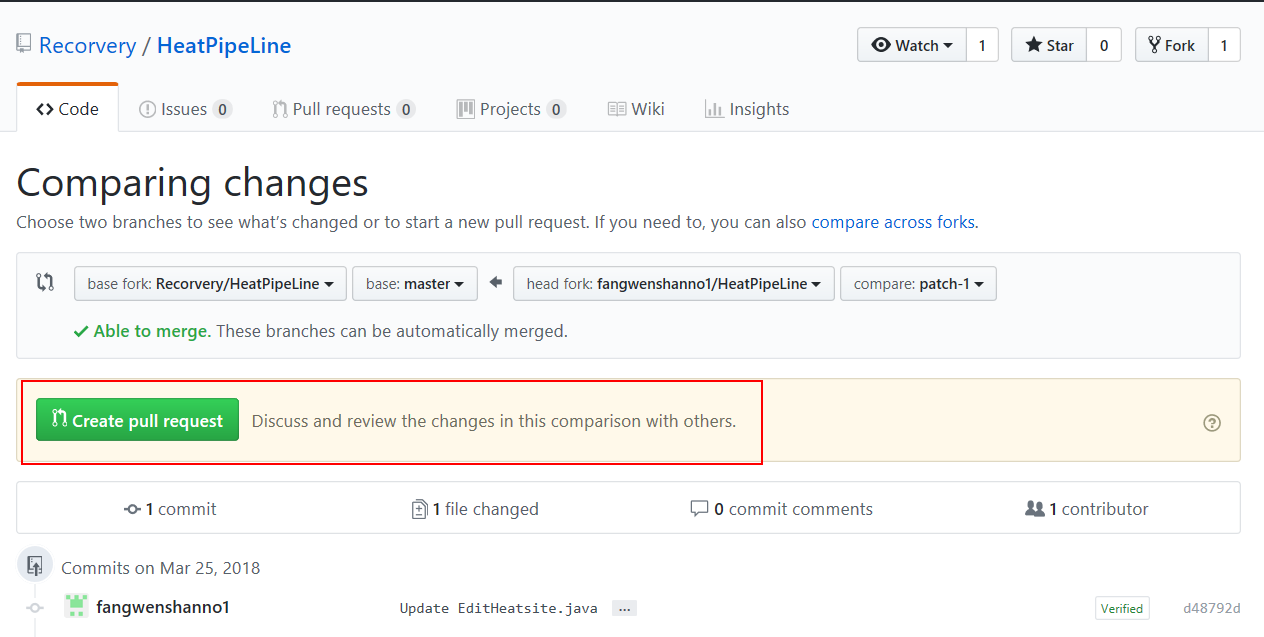
①搜索感兴趣的项目



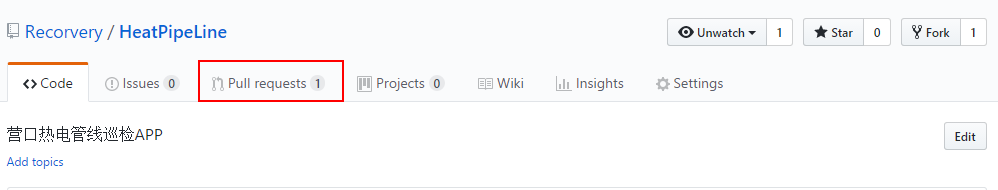
②fork到自己的远程库，fork到自己的github后，自己可以进行编辑！



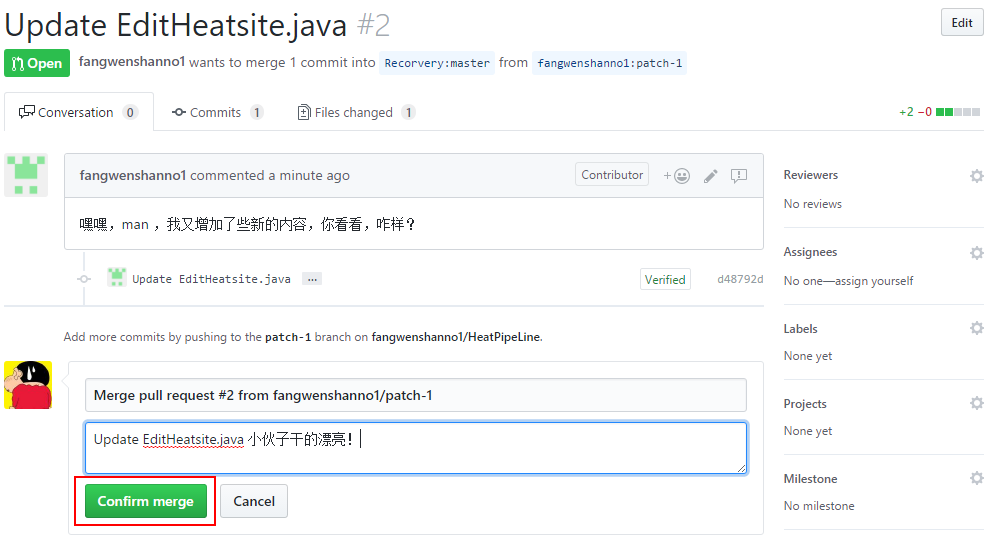
③将自己做的pull request给原作者，以等待原作者采纳！



④原作者查看后，执行合并操作



⑤确认没有冲突后，执行合并操作！



### 3.8 总结



# 七、在 idea 中使用 git 和 github

## 1. 给idea 配置 git

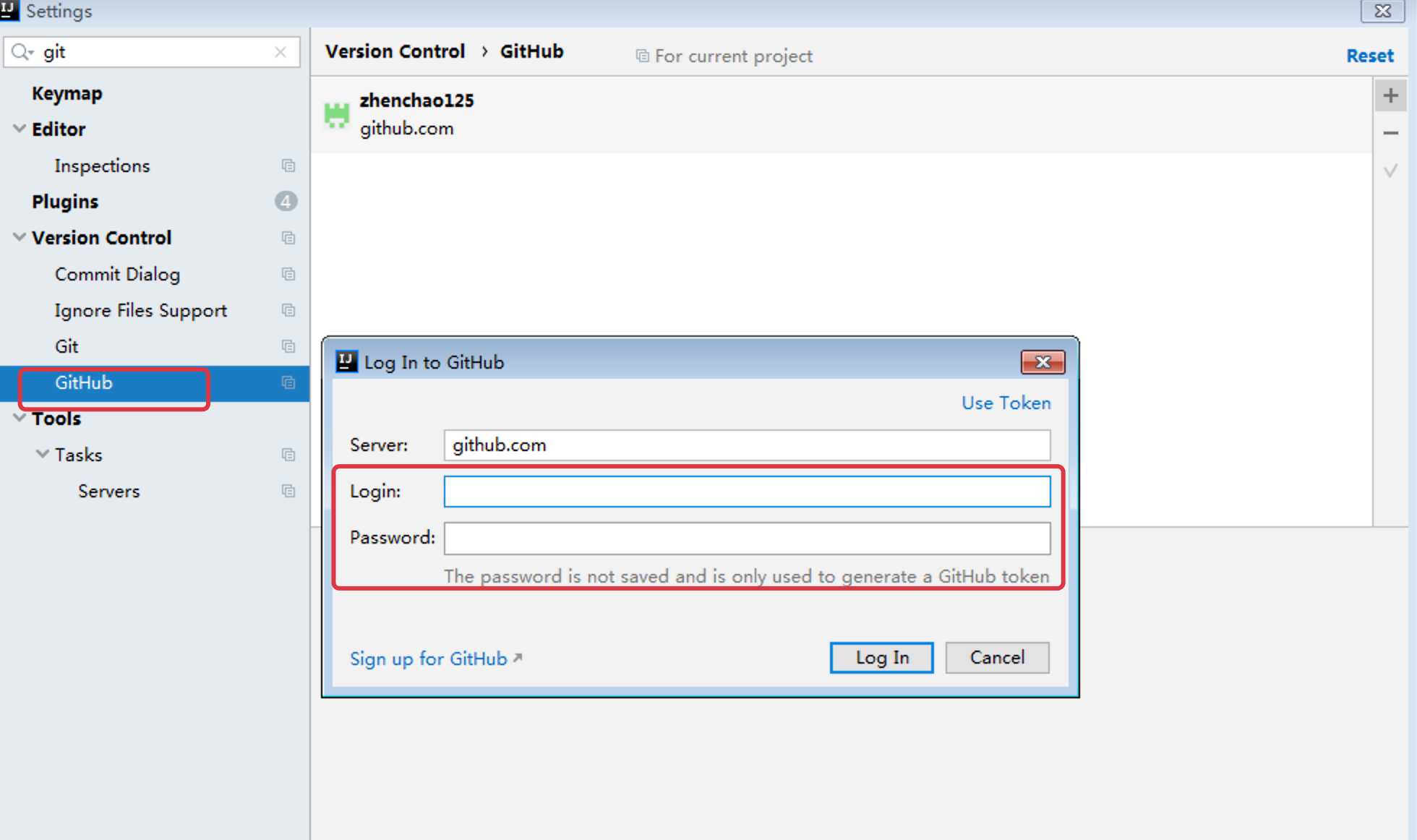
点击 test, 如果显示 ok,则 idea 自动找到了当前操作系统安装的 git 程序.否则需要手动指定 git.ext 的位置





## 2. 给 idea 配置 github

为了方便代码可以托管到 github, 需要配置你的 github 的账户信息.



## 把项目推送到 github

会在本地初始化 git 仓库然后推送到 github

